College of Computer & Information Science
Northeastern University
CS5800: Algorithms

Spring 2011
Handout 10
3 March 2011

Quiz 2

Name: _____

(10 points) Selection in Bitonic lists

You are working for an investment firm that prides on its analysis of the past. (You have heard this before!) You have been regularly asked to analyze historical data, which are being updated every minute and hence result in huge lists (of the order of several millions). One of the problems you repeatedly face is to find the minimum, maximum, or median value in such a list.

Of course, you know that the maximum, the minimum, or, in fact, element of any given rank, in any list can be computed in linear time. But since the lists are huge, even linear time is prohibitively large. Fortunately, most of the lists you analyze are *bitonic*: that is, the list is composed of two contiguous (possibly empty) segments, the first is an increasing segment and the second is a decreasing segment.

Formally, an array $A[1 \ldots n]$ is bitonic if there exists an integer k , $1 \leq k \leq n$ such that for $1 \leq i < k$, $A[i] < A[i+1]$, and for $k \leq i < n$, $A[i+1] < A[i]$. For example, the lists $[2, 4, 6, 7, 5, 3, 2]$ and $[1, 2, 5, 6, 8]$ are bitonic while the list $[9, 10, 12, 10, 7, 8]$ is not bitonic.

For each of the following parts, *give an efficient algorithm for finding the desired element of an n-element bitonic list, and state the worst-case running time (as a function of n) of your algorithm.* Note that the running-time is measured by the number of basic arithmetic and logical operations and the number of list entries that your algorithm accesses in the worst case. For each part, make your algorithm as efficient as you can, in terms of its worst-case running time. (You need not prove the correctness of your algorithm or give an analysis of its running time.)

(a) (6 points) Maximum element.

Answer: A bitonic array is composed of two contiguous segments, the first segment is increasing and the second is decreasing. Suppose we denote this array as A, then our algorithm can be described as follows:

1) Calculate the median of the input arrays A, which is $A[n/2]$.
2) If $A[n/2]$ is greater than both of its left and right elements in the array. return $A[n/2]$.
3) If $A[n/2]$ is greater than its left neighbor and smaller than its right, then max is present in $A[n/2]$ to $A[n-1]$
4) If $A[n/2]$ is smaller than its left neighbor and greater than its right, then max is present in $A[0]$ to $A[n/2]$
5) Repeat the above process from step 2 to 4 until we find the maximum of the whole array.

Every time we shrink the problem size into half of the previous, so the total running time of this algorithm will be $O(\log n)$ in the worst case.

(b) (2 points) Minimum element.

Answer: Since in a bitonic array, the first is an increasing segment and the second is a decreasing segment. $A[0]$ and $A[n-1]$ are the smallest numbers of each segment respectively. So the minimum of the whole list is the minimum of $A[0]$ and $A[n-1]$.

(c) (2 points) Median element.

Answer: We can use random selection to get the median, but the worst case will be $O(n^2)$. Also we can preprocess this array by sorting, then we can get the median by constant time, however best sorting algorithm will still take $O(n \log n)$ time. We can also $O(n)$ deterministic selection algorithm, but none of these algorithms take advantage of the bitonic property.

Here is a much faster – $O(\log n)$ time – algorithm. We will use the solution of problem(a) to get the maximum of a bitonic array. Since the first is an increasing segment and the second is a decreasing segment, we can consider them as two sorted arrays divided by the maximum. The problem to find the median of a bitonic array will be converted to find the common median of two sorted arrays.

Suppose we denote these two sorted arrays as A_1 and A_2, then our algorithm can be described as follows:

1) Calculate the medians m_1 and m_2 of the input arrays A_1 and A_2 respectively.
2) If m_1 and m_2 both are equal then we are done. return m_1 (or m_2)
3) If m_1 is greater than m_2, then median is present in one of the below two subarrays.
   a) From first element of A_1 to m_1.
   b) From m_2 to last element of A_2.
4) If m_2 is greater than m_1, then median is present in one of the below two subarrays.
   a) From m_1 to last element of A_1.
   b) From first element of A_2 to m_2.
5) Repeat the above process until size of both the subarrays becomes 2.
6) If size of the two arrays is 2 then use below formula to get the median.
   Median = $(\max(A_1[0], A_2[0]) + \min(A_1[1], A_2[1]))/2$.

Every time we shrink the problem size into half of the previous one, so the total running time of this algorithm will be $O(\log n)$. Note that this algorithm is very similar to Problem 2.22 of text that we covered in Recitation 2.